ГБОУ Гимназия № 1505
«Московская городская педагогическая гимназия – лаборатория»

Диплом

Применение концепции автономных агентов для реализации алгоритма деятельности космической станции

автор:
Манаева Ангелина, 10 класс «А»

руководитель:
Наумов А.Л.

Москва

2014-2015

Введение

С развитием игровой индустрии, популярности поисковых машин и интернет ботов, программисты все чаще обращаются к концепции автономных агентов - программных модулей, для которых окружающий программный мир выступает своеобразной средой, под воздействием которой они принимают те или иные решения.

Скажем, при работе по созданию примитивного искусственного интеллекта для рядовой стратегической игры на планшете или компьютере, существенно проще разбить общее поведение системы на отдельные модули (танки, самолетики, герои, …), так называемые, автономные агенты и писать алгоритмы для них. Важно, что для каждого такого агента, компьютерный уровень представляет собой неизвестную величину, то есть, агент-программа “живет” в виртуальной реальности так же, как персонаж пользователя.

В настоящем дипломе, я расскажу об автономных агентах и опишу модель агента на языке Processing[1]. Эта модель будет играть роль робота, который будет существовать в некоторой части вселенной и будет поддерживать жизнедеятельность космической станции. Для этого мне будет необходимо изучить литературу по автономным агентам, познакомиться с Processing и в нем построить модель агента.

Данная тема диплома и результат исследования могут быть полезны, так как исследование на основе программы, созданной в ходе написания дипломной работы, может быть использовано для написания игр на платформе JavaScript, а также текст моей работы может быть полезен тем, кто хотел узнать об автономных агентах и способе их программирования в языке Processing.

[1] Processing - открытый язык программирования, основанный на Java.

Глава 1
Основные принципы программирования автономных агентов

Данная глава будет являться вводной и в ней мы более подробно познакомимся с автономными агентами, как с одним из типов программ, а также рассмотрим принцип создания программы на основе написания агентов для отдельного объекта и последующего соединения их воедино.

1.1 Автономные агенты

Итак, что же такое автономные агенты?[2] Обычно, при программировании мы создаем программы, которые работают строго по тем направлениям, что мы им задали, и по сути, их поведение можно легко предсказуемо: программа запускается, получает входные данные, последовательно выполняются заранее написанные команды, выводится конечный результат. До следующего запуска извне (если он вообще состоится) она пребывает в состоянии спящего режима или ожидания, и не имеет отношения к реальному времени..

Главным отличительным признаком автономного агента от обычной программы является то, что он функционирует (запускается, приостанавливается, возобновляется, завершается, запускается вновь) в режиме реального времени. Еще важный признак автономного агента состоит в том, что он может оценивать результаты своего выполнения и менять свои действия в будущем с учетом прошлых недочетов или, наоборот, успехов. Таким образом, агенты – это намного более гибкая конструкция, чем привычные нам программы.

В 1996 году было предложено следующее определение агента: **автономный агент** – это система, находящаяся внутри окружения и являющаяся его частью, воспринимающая это окружение (его сигналы) и воздействующая на окружение для выполнения собственной программы действий[3].

---

[2] Лекция 10: Распределенные интеллектуальные системы на основе агентов. - – Электрон. данные. – – Режим доступа: http://www.intuit.ru/studies/courses/1146/238/lecture/6154 ,свободный. – Данные соответствуют 04.05.15.

[3] Такое определение принадлежало С.Франклину и А.Грэссеру

К числу наиболее характерных свойств автономных агентов следует отнести[4] :

- *автономность* - способность к самостоятельному оценке происходящего, формированию целей и переход в определенное состояние, в зависимости от ситуации;

Как и все последующие примеры, этот будет основан на сериях симулятора футбольных игр - FIFA[5]. Рассмотрим пример с вратарём. Он находится в режиме ожидания, если мяч достаточно далеко от него, если же мяч близко, то вратарь, реагируя на любую опасную ситуацию, решает что ему делать - готовиться перехватывать мяч, отражать атаку или нейтрализовать удары. Важно, что деятельность вратаря - результат внутренних алгоритмов, а не результат прямых указаний системы.

- *реактивность* - взаимодействие с окружающим миром, получение информации и ее дальнейший анализ, извлечение из него знаний, которые пригодятся в дальнейшем;

Скажем, при оценке того, как поступить в том или ином случае, вратарь оценивает расстояние до мяча, направление движения мяча и другие параметры.

- *целенаправленность* - означает, что у агента имеется определенная цель и его поведение меняется в зависимости от нее;

Цель вратаря не пропустить гол. В зависимости от поступающей информации, вратарь будет смещаться в ту или иную сторону.

- *непрерывное функционирование* - постоянно нахождение в состоянии выполнении неких действий;

Движение, поиск цели, взаимодействие с другими игроками - все это происходит у всех агентов-игроков вне зависимости от того, какой игрок владеет мячом в данный момент.

- *коммуникативность* – общение с другими агентами;

Футболисты и вратарь общаются между собой посылая пакеты данных: в тот момент когда вратарь выкидывает мяч на поле, он уже знает, где находится какой игрок.

- *обучаемость (адаптивность*) – способность корректировать свое поведение, основываясь на предыдущем опыте;

В одной из последних версий игры точность выброса определялась навыками виртуального вратаря. Неверные решения, как например слишком сильные или слишком слабые броски, приводили бы к печальным результатам - отбору мяча командой противника. Каждый такой неудачный “опыт” менял

---

[4] А.Н. Швецов, “ Агентноориентированные системы: от формальных моделей к промышленным приложениям “ – Электрон. данные. – – Режим доступа: http://www.ict.edu.ru/ft/005656/62333e1-st20.pdf ,свободный. – Данные соответствуют 04.05.15.

[5] FIFA - серия симуляторов футбола, которая разрабатывается студией EA Canada.

параметры внутри агента - что приводило к тому, что в дальнейшем агент более адекватно справлялся с поставленной задачей.

- *мобильность* – способность менять свое положение;

Реалистичное передвижение игроков по полю - основная составляющая игры.

- *гибкость* – действия агента не примитивны и не очевидны;

Деятельность агента подчинена ряду алгоритмов, каждый из которых сравнительно прост (пойди, посмотри, ударь, …) однако переключение алгоритмов в соответствии с внешней средой рождает достаточно сложную поведенческую линию.

- *индивидуальность* – наличие персональных свойств.

Например, в FIFA есть возможность создавать игрока, который будет в ряде качеств похож на пользователя[6]. Параметры “схожести” будут влиять на те или иные коэффициенты внутри алгоритма.

Использование автономных агентов удобно тем, что оно позволяет, не зная определенного способа решения или необходимых параметров, обойтись минимумом усилий, создав только один прототип агента, которых затем запускается в среду и достаточно эффективно выполняет анализ информации, адаптируясь и постепенно достигая той цели, которая была заявлена.

Скажем, в случае с вратарем, сложно изначально запланировать все возможные ситуации на поле, куда проще и реалистичнее - предоставить программе возможность самой решать, что делать, с тем, что бы ее цель была достигнута наиболее эффективно.

Давайте на примере моделирования отдельных птиц, предложенным Крейгом Рейнольдсом, рассмотрим подробнее применение автономных агентов. С помощью схожего моделирования Крейг в дальнейшем создал модели поведения стаек рыб и муравьев.

1.2 Принцип моделирования движения К.Рейдольдса

Для моделирования любого автономного агента, в том числе птицы, нам потребуется решить ряд проблем, из которых две - наиболее важны:

1) как моделировать движение
2) какие алгоритмы заложить в поведение агента

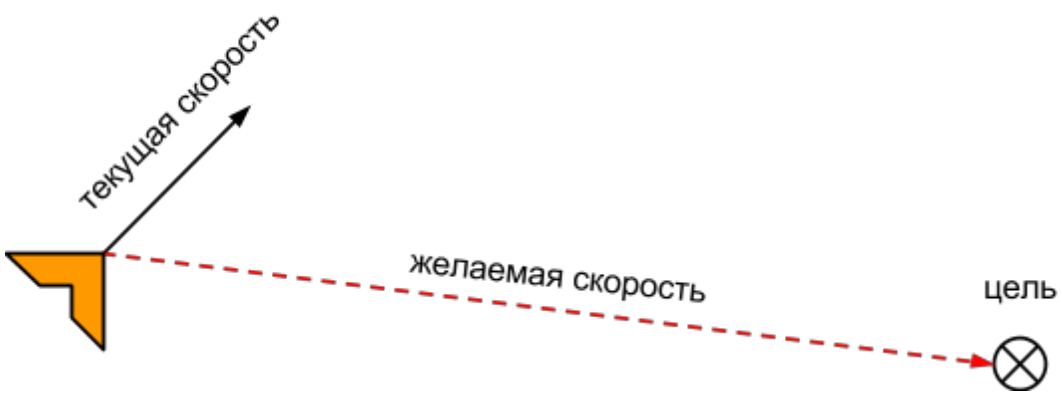
[6] масса, рост, максимальная скорость и так далее

В данном параграфе мы рассмотрим только первый вопрос, адекватное моделирование передвижения агента, тогда как алгоритмам будет посвящена вторая глава[7]

Так как передвижение птиц похоже на движение космических объектов[8] при программировании движения автономных агентов можно опереться на описание К.Рейнольдсом движения идеальных средств передвижения[9]. Итак, К.Рейнольдс выделяет 3 этапа движения - выбор цели, приложение силы и передвижение.

1. Выбор цели - объект понимает, где находится конечная цель его движения (либо локализуя ее координаты, либо устанавливая вектор перемещения между текущим положением объекта и конечным - “желательным” положением)
2. Приложение силы - вектор текущей скорости объекта должен быть изменен так, чтобы агент двигался в нужном направлении
3. Передвижение - изменение координаты агента

Предположим, что мы рассматриваем ситуацию, когда небольшой космический кораблик (далее “дрон”) должен достичь некоторой точки в пространстве. Рассмотрим, как программно этого достичь.



(Рис. 3а)

Как видно из рисунка (Рис. 3а), скорость дрона направлена таким образом, что он не в состоянии достичь цели. Для того, чтобы он оказался в

[7] по той причине, что алгоритмы жестко привязаны к конкретной задаче, тогда как движение может быть использовано в разного рода моделях

[8] движение реактивное: физика во многом схожа

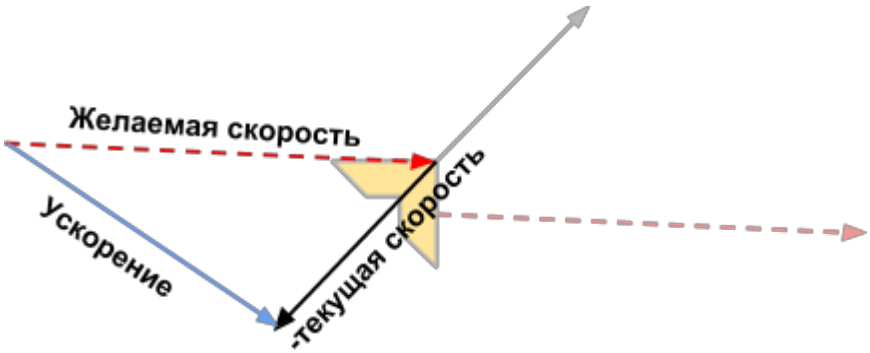
[9] Они являются идеальными из-за того, что они никак не связаны с реальными механизмами транспортных средств: мы не знаем спецификаций двигателей, аэродинамических особенностей и так далее

заданном месте пространства, требуется чтобы его скорость была направлена так, как показано красной стрелкой.

Для этого требуется сделать следующие операции.

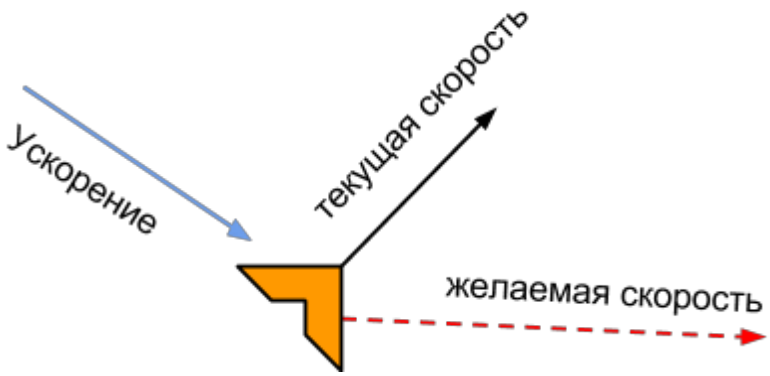
1. узнать, направление текущей скорости дрона
2. вычислить разницу между текущей и желаемой скоростями таким образом получив необходимое ускорение

(чтобы получить численную величину ускорения нам необходимо из вектора желаемой скорости вычесть вектор текущей скорости,а т.е. прибавить вектор текущей скорости с противоположным знаком, выполняется по правилу сложения векторов. (Рис. 3б)



(Рис 3б)

3. уменьшить величину ускорения до некоторой оптимальной величины[10]
4. сложить приведенное ускорение с текущей скоростью (Рис 3в)



(Рис 3в)

5. если текущая скорость получилась больше чем некоторое заданное число - уменьшить ее до заданного[11]

[10] В противном случае объект будет достигать заданной цели каждую следующую итерацию

[11] в принципе, в космосе возможно достижение любых скоростей, однако для нашей симуляции мы ограничим возможности дрона, в противном случае он будет тратить слишком много времени на торможение при “пролете” цели

6. сложить вектора скорости и текущего положения

Наша задача, как считает Рейнольдс, состоит в перемещении автономного персонажа в наиболее "реалистичной манере". То есть, если бы мы просто сказали, что наши координаты эквиваленты координатам цели, то дрон переместится бы туда мгновенно, однако это было бы не реалистично.

Мы же пытаемся добиться иного эффекта - как будто дрон, управляя своим передвижением, приходит к своей цели различными способами, и, изменяя тягу (ускорение), моделирует изменение скорости. Другими словами, чем больше мощность двигателя дрона, тем больше может быть ускорение дрона. На рисунке 3г приведены траектории дрона с большим и маленьким ускорениями (Рис.3г)

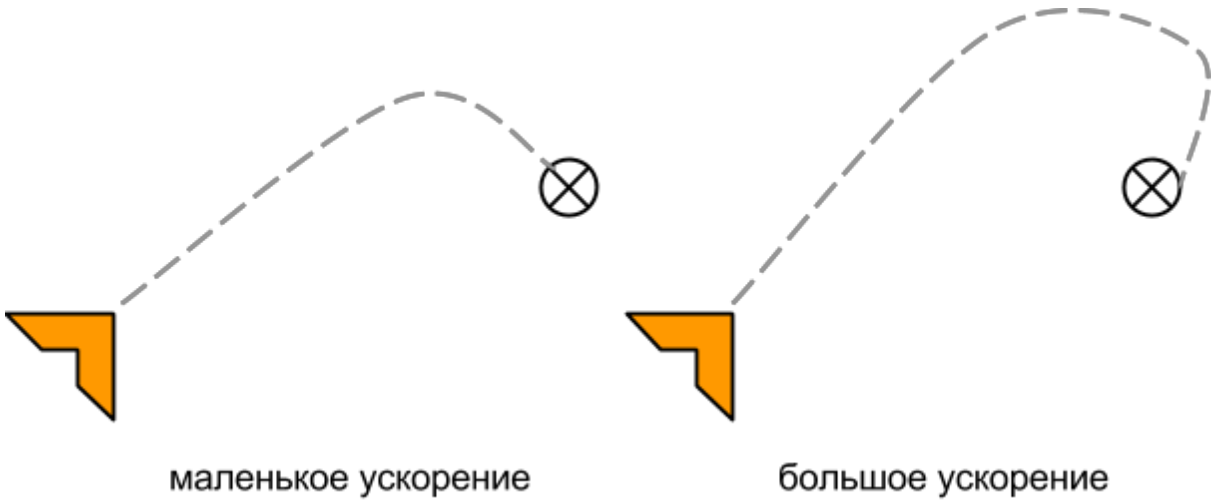
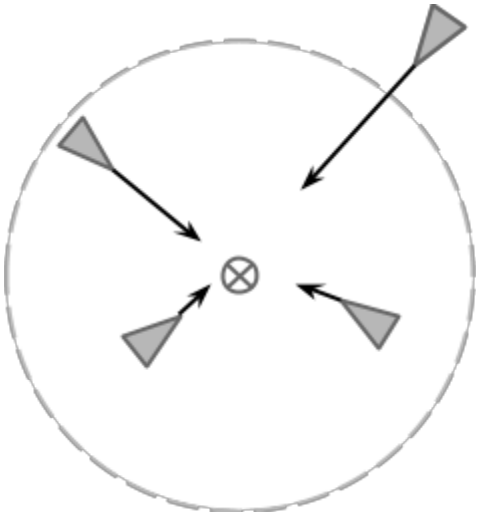


Рис. 3 г)

Теперь, разобравшись с передвижением, мы должны рассмотреть ситуацию с прибытием дрона на астероид. Рейнольдс описывает это следующим образом:

“Представим себе круг вокруг цели с заданным радиусом. Если дрон находится в пределах этого круга, это начинает замедляться на краю круга, его скорость постепенно уменьшается так, что когда он оказывается ровно в центре круга его скорость равна 0”. (Рис.4)

Такое поведение реализовано следующим образом:



(Рис 4)

1. у дрона существует переменная, в которую занесено расстояние между ним и его целью, которое вычисляется через формулу длины вектора, где (x1, y1) - координаты дрона, а (x2, y2) - координаты цели
2. также у дрона есть переменная, которая отвечает за то расстояние от цели, на котором он должен начать замедляться
3. при ситуации, когда переменная1 < переменной2, наш дрон постепенно начинает замедляться пропорционально уменьшению расстояния. Это происходит за счет изменения параметра максимальной скорости. Чем ближе дрон к цели, тем меньше может быть его максимальная скорость. При достижении центра цели его скорость дрона будет равна нулю.

Глава 2
Моделирование автономного агента

2.1 Сюжет программы и целеполагание агента

Так как нередко, автономные агенты используются для реализации поведения объектов в компьютерной игре, для нашего исследования был выбран следующий сюжет:

“Вы попадаете в маленькую часть Вселенной, в которой происходит непонятное движение, а также наблюдаются следы неизвестной Вам жизни, которая отображается на радарах космических кораблей и иных станций как “Враждебный Объект”. Изначально, события, которые Вы наблюдаете, происходят миллиарды лет назад. Перед Вами планета/база/иное космическое тело (Вам этого не известно). Спустя какое-то время около него начинают появляться множества маленьких роботов, которые по какой-то причине начинают поддерживать жизнедеятельность данного космического объекта… ”

Автономные агенты в данном случае будут играть роль дронов, которые были упомянуты выше. Как нам уже известно из сюжета, дроны поддерживают жизнедеятельность космического тела (далее - Базы). Для этого они летят на Астероид, на котором содержаться ресурсы, необходимые Базе.

Перед тем, как отправится на помощь Базе, дрон самостоятельно принимает ряд решений - необходимы ли базе ресурсы, куда ему за ними лететь, сколько необходимо взять с собой, а также не забывает о себе, так как у дрона есть несколько параметров, которые отвечают за его существование: количество топлива, здоровье, а так же масса, от которой зависят его максимальная загруженность и скорость. Для каждого дрона эти параметры уникальны, так как зависят от его массы, которая устанавливаются случайным образом. .

2.2 Поведение агента

2.2.1 Общая логика поведения

Для алгоритмизации поведения дронов, была создана величина - “state”. Она определяет его состояние в тот или иной момент. Каждое его состояние и его изменения описаны в таблице, указанной ниже. Если мы взглянем не эту схему, то видно, что конца у нее нет. А конец, а то есть смерть Дрона, происходит тогда когда переменная, отвечающая за его исчезновения,

возвращает 0. Это проверяется каждый шаг, поэтому она не указана в целях удобства восприятия.

Таблица 1

Состояния автономного агента и его соответствующее поведение		
текущее состояние	**пояснение**	**переход в новое состояние**
state 0	Спящий режим - если база не нуждается в ресурсах; Если база нуждается в ресурсах	state 1
state 1	Поиск ближайшего астероида для сбора ресурсов; Алгоритм перебора астероидов (дрон ищет ближайший из всех возможных)	state 1.2
state 1.2	Диагностика состояния астероида; Проверка степени “свободы” выбранного астероида ; если он свободен (то есть никто не планирует его добывать”), то происходит установка цели (ближайший и свободный астероид) и у астероида отнимается единица свободы;	state 2
state 2	По прибытии на астероид Дрон начинает набирать ресурсы, при этом его здоровье тратится. Это происходит до тех пор, пока он не накопит максимальное количество ресурсов ;	state 3
state 3	Дрон набирает достаточное ресурсов. Покидая астероид, он возвращает ему единицу свободы. Дрон следует на Базу для передачи ресурсов ;	state 6
state 6	Максимальное количество ресурсов, единица свободы возвращена астероиду - дрон готов к состыковке с Базой и дальнейшей передачи ресурсов ;	state 4
	Передача всех(!) собранных ресурсов Базе, также	state 0

state 4	происходит восстановление здоровья и пополнение запасов топлива до их максимального значения ;	

Если внимательно посмотреть на перечень состояний, то можно заметить, что действия выполняемые дроном можно свести к следующим:

- перемещаться
- находить ближайший не занятный астероид
- добывать астероид (выгружать запасы на базе)

2.2.2 Алгоритм выбора астероида

Из перечисленных выше действий, неописанным остался только выбор ближайшего астероида. Заметим, что в рамках заданных условий, дрон не имеет общей таблицы с астероидами и расстояниями до них, так же “программа в целом” не может сообщить ему куда лететь, это решение должно быть принято дроном на основе полученной им информации. То есть дрон будет поочередно запрашивать расстояние до астероидов (как будто у него есть сенсор - дальномер), и далее искать оптимальный.

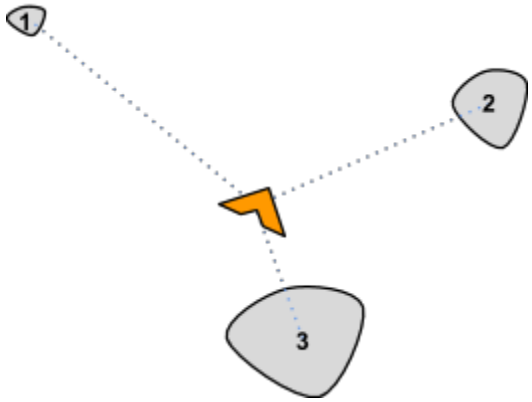
Помимо просто поиска ближайшего, нам показалось более реалистичным, что только ограниченное количество дронов могут добывать один астероид (в нашем случае 3), следовательно потребовалось адаптировать алгоритм поиска так, чтобы находить не просто ближайший, а ближайший не занятый.

Итак, теперь давайте подробнее разберем то, как все таки дрон выбирает себе наиболее подходящий астероид. Как уже было сказано выше , для экономии времени нам необходим самый ближайший. Для того, чтобы найти его нам необходимо совершить следующее:

1. изначально в переменную, которая отвечает за минимальное расстояние до астероида, дрон заносит самое большое расстояние, а то есть диагональ экрана ;

Далее в дрон запрашивает координаты x и y для каждого астероида (“сенсор - дальномер”) ;

2. вычисляем расстояние между дроном и каждым астероидом через формулу длины вектора, где (x1, y1) - координаты дрона, а (x2, y2) - координаты астероида. В данном случае у нас есть 3 расстояния, т.к. количество астероидов равно 3 (Рис. 5)



(Рис 5)

3. сравниваем каждую полученную раннее величину с числом, записанным в переменной минимального расстояния. Если эта величина меньше заданной, то минимальным расстоянием становится это расстояние , и в координаты цели записываются координаты данного астероида. Если же величина больше ,то ничего не меняется ;

4. в итоге, выходит, что расстояние 1 > расстояния 2, а расстояние 2 > расстояния 3. Следовательно, расстояние3 минимально и в данном случае дрон полетит на астероид под номером 3

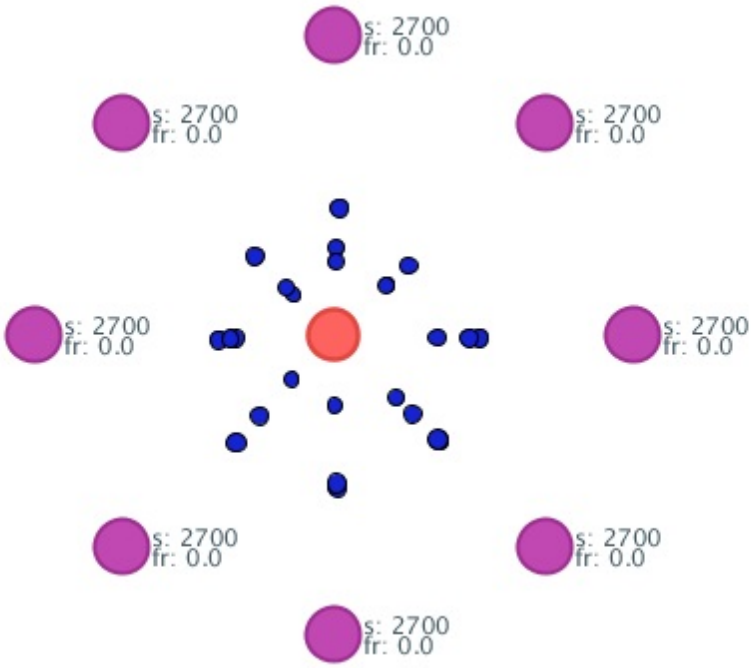
2.2.3 Алгоритм реализации “занятости” астероида

После того, как цикл завершился, в координатах цели остаются x и y того астероида, чье расстояние между ним и дроном минимально, а, следовательно, он и является ближайшим астероидом. Как уже было упомянуто выше, у астероида есть так такая характеристика, как Степень Свободы. Она определяет количество дронов, которых астероид может вместить в себя. Когда дрон наконец нашел себе ближайший, он проверяет Степень Свободы данного астероида. Если она больше нуля, то он посылает ему сигнал о том, что теперь этот астероид - его цель, и Степень Свободы астероида понижается. Когда, набрав достаточное количество ресурсов, дрон покидает астероид, Степень его Свободы вновь повышается.

2.2.4 Итог реализации программы автономного агента

Для того, чтобы лучше понять содержание программы, ознакомимся с рядом скриншотов сделанных вовремя ее работы.

На рисунке 6 изображено начало жизнедеятельности Базы, на ней нет ресурсов ... Дроны, заметив недостаток ресурсов на Базе, устремляются к ближайшим свободным Астероидам.

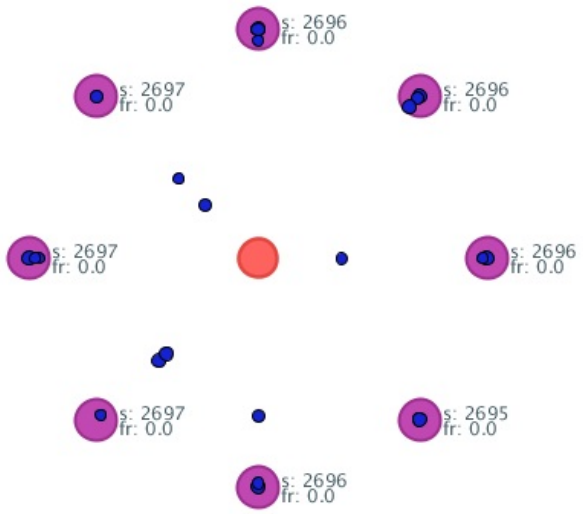


(Рис. 6)

Рис.7 - собрав достаточное количество ресурсов, дрон направляется на Базу. Здесь также можно заметить некоторое различие между окрасками дронов (чем он темнее, тем больше рейсов на астероиды он совершил).

(Рис. 7)

Рис.8 - при накоплении ресурсов, находясь на Астероиде, дроны теряют здоровье. При его эквивалентности нулю дрон погибает . Дабы не приостанавливать поддержку жизнедеятельности, при смерти одного дрона База выпускает нового дрона, которые тут же направляется на добывание ресурсов, которое было прервано его сотоварищем.



(Рис. 8)

Итак, давайте подведем итоги. Создан дрон, который

1) выполняет роль автономного агента, и он является объектом, содержащим свои персональные параметры. Основным и главным из них является масса - от нее зависит все: количество топлива, здоровье, максимальная загруженность и скорость. Во всех случаях зависимость того или иного параметра является прямой пропорциональностью - чем больше, тем больше. Исключением является скорость, зависимость скорости и массы обратна - чем больше масса, тем медленнее объект набирает скорость и тормозит.

2) содержит алгоритм поведения, заданный таблицей 1

3) может перемещаться (то есть, создан алгоритм перемещения)[12]

То есть мы имеем объект со своими индивидуальными параметрами - автономного агента. В таблицах ниже приведены параметры дронов, которые достигли определенного количества прибытия на астероид 10 единиц и более.

[12] см. 1.2 Принцип моделирования движения К.Рейдольдса

Следует заметить, что значительное количество дронов погибает на пути на астероид или обратно, то есть, те, которые могут совершить 10 и более полетов - как правило наиболее приспособленные.

Благодаря им мы можем узнать коэффициент зависимости здоровья и топлива от главного параметра - массы. Исходя из данных, приведенных в таблице, получается, что здоровье относится к массе как 3 : 1, а топливо к массе как 5 : 3 . Эти отношения дают нам возможность создать наиболее подходящих дронов, а то есть таких, которые дольше всего находятся в работоспособном состоянии.

Заметим на полях, что коэффициент “старения” дрона[13] выбран на данном этапе произвольно, и может быть изменен в согласии с реальными данными корабля.

Таблица 2

Выборка 1										
Параметр	Значение									Ср. значение
Fuel	16	13	13	15	16	15	16	14	16	15
Health	39	28	31	35	39	41	30	39	40	36
Mass	13	7	11	11	12	13	9	14	14	12

Таблица 3

Выборка 2										
Параметр	Значение									Ср. значение
Fuel	16	13	13	15	16	15	16	14	16	15
Health	39	28	31	35	39	41	30	39	40	36
Mass	13	7	11	11	12	13	9	14	14	12

[13] Величина показывающая насколько будет уменьшаться количество топлива дрона при пролете единицы пространства, а так же, как уменьшается здоровье при добыче астероида

Таблица 4

Выборка 3										
Параметр	Значение									Ср. значение
Fuel	18	14	14	19	16	15	17	14	14	16
Health	37	32	32	39	36	34	38	35	36	35
Mass	13	10	11	14	11	12	12	12	11	12

Заключение

В ходе исследовательской работы я достигла своей главной цели – создание программы, в которой содержатся основные объекты изучения - автономные агенты . Для достижения этой цели я выполнила следующие задачи:

- изучила литературу по разделу «автономные агенты»;
- изучила литературу по такому языку программирования как Processing, основный на Java;
- построила модель автономного агента, пользуясь языком программирования Processing, обрисовав ее в космической тематике

В первой главе исследовательской работы был изложен основной материал по теме «автономные агенты»: было изучено основное понятие автономных агентов, приведены их свойства, а также принцип моделирования К.Рейнольдса, который был взят за основу в практической части. Во второй главе была разобрана программа, написанная в ходе исследования : ее сюжет, подробное описание функций Дрона и его поведение.

Продуктом исследования является программа, которую Вы можете найти в Приложении. Эта программа содержит в себе модель автономного агента и в данном случае автономные агенты использованы для реализации алгоритма деятельности космической станции. Эту программу можно далее использовать для написания наиболее масштабных программ на космическую тематику или же как наглядную модель автономных агентов.

Список литературы

1. Официальный сайт языка программирования Processing. // http://processing.org/ Ссылка действительна на 27.04.15.
2. Daniel Shiffman. “Nature of Code”. // http://natureofcode.com/book/ Ссылка действительна на 27.04.15.
3. Craig Reynolds. Home page of Craig W. Reynolds. // http://www.red3d.com/cwr/
   Ссылка действительна на 27.04.15.
4. Mitchel Resnick. “ Turtles, Termites, and Traffic Jams “. // http://www.melaniecrean.com/collaborativefutures/wp-content/uploads/2013/08/Resnick_TurtlesTermitesTraffic.971.pdf Ссылка действительна на 27.04.15.
5. Casey Reas and Ben Fry. Getting Started with Processing. // http://it-ebooks.info/book/244/
   Ссылка действительна на 27.04.15 , p129 - 138
6. Е. Варфоломеева, Перевод статьи Крейга Рейнольдьса. http://window.edu.ru/resource/251/51251/files/2006_2_78-84.pdf // Ссылка действительна на 29.04.15.
7. Лекция 10: Распределенные интеллектуальные системы на основе агентов. http://www.intuit.ru/studies/courses/1146/238/lecture/6154 // Ссылка действительна на 29.04.15.
8. Сергей Бобровский, “Досье искусственного интеллекта. Часть 2 “// http://xn--90anaif2c.xn--p1ai/ai_program.htm
   Ссылка действительна на 04.05.15.
9. А.Н. Швецов, “ Агентноориентированные системы: от формальных моделей к промышленным приложениям “ // http://www.ict.edu.ru/ft/005656/62333e1-st20.pdf
   Ссылка действительна на 04.05.15.

Приложение :

1) Программа целиком

```
ArrayList <Dron> drons = new ArrayList <Dron> ();
ArrayList <Asteroids> wPlace = new ArrayList <Asteroids> ();

int dronS = 25;
int astS = 8;
int sw = 0;
float _x;
float _y;
int part1 = 150;
int part2 = 106;

Base MainShip;


void setup() {
  size(600, 600);

  //adding base - mainship, which makes some vital functions and activity .
  MainShip = new Base();

  /* Without resourses MainShip's vital functions will be over, so MainShip needs someone to drag these resources on it.
   Due to the fact that there are two types of resources - water and ... - MainShip has two types of robots - Drons and Bots for water and ... respectively .
   And there are two tipes of places, where these resourses are - Asteroids for water and Meteors for ...  .
   */


  //adding place, where Asteroids are .
  //max - 2000 res
  for (int i = 0; i < astS; i = i + 1) {

    if (i == 0) {
      _x = MainShip.getX() + part1;
      _y = MainShip.getY();
```

```
    }
    if (i == 1) {
      _x = MainShip.getX() + part2;
      _y = MainShip.getY() + part2;
    }
    if (i == 2) {
      _x = MainShip.getX() - part1;
      _y = MainShip.getY();
    }
    if (i == 3) {
      _x = MainShip.getX() + part2;
      _y = MainShip.getY() - part2;
    }
    if (i == 4) {
      _x = MainShip.getX();
      _y = MainShip.getY() + part1;
    }
    if (i == 5) {
      _x = MainShip.getX();
      _y = MainShip.getY() - part1;
    }
    if (i == 6) {
      _x = MainShip.getX() - part2;
      _y = MainShip.getY() - part2;
    }
    if (i == 7) {
      _x = MainShip.getX() - part2;
      _y = MainShip.getY() + part2;
    }


    wPlace.add(new Asteroids (_x, _y, 2700));
  }

  //adding Drons
  for (int i = 0; i < dronS; i = i + 1) {
    drons.add(new Dron (MainShip.getX(), MainShip.getY(), random(5, 15), random(1,
10), random(1, 15) ));
  }
}
```

```
void draw() {
  background(#FFFFFF);
  fill(247, 107, 107);

  //drawing Base
  MainShip.script();


  //drawing Asteroids
  for (Asteroids watty : wPlace) {
    watty.display();
  }

  //drawning Drons
  for (int i = 0; i< drons.size (); i++) {
    float p = 0;
    Dron myDron = (Dron) drons.get(i);

    // find the nearest AsteroidsPlace
    if (myDron.getState() == 1) {
      for (int n = 0; n < wPlace.size (); n++) {
        Asteroids AsteroidsPlace = wPlace.get(n);
        myDron.LocateWater(n, AsteroidsPlace.getX(), AsteroidsPlace.getY(),
AsteroidsPlace.getFreedom(), AsteroidsPlace.isDead);
      }

      for (int n = 0; n < wPlace.size (); n++) {
        Asteroids AsteroidsPlace = wPlace.get(n);
        if ((myDron.nearestPlace.x == AsteroidsPlace.getX()) &&
(myDron.nearestPlace.y == AsteroidsPlace.getY())) {
          AsteroidsPlace.loseFreedom(1);
        }
      }
      myDron.setState(12);
    }
    //  Dron tells Asteroid, that asteroid is being eaten, while Dron is on it.
    if (myDron.getState() == 2) {
      for (int n = 0; n < wPlace.size (); n++) {
        Asteroids AsteroidsPlace = wPlace.get(n);
        if (dist (myDron.getX(), myDron.getY(), AsteroidsPlace.getX(),
AsteroidsPlace.getY()) < 50) {
          AsteroidsPlace.isBeingEaten(myDron.getPart());
```

```
      }
     }
    }
    //  Dron says Asteroid, that asteroid is not being eaten, because Dron goes away
    if (myDron.getState() == 3) {

     for (int n = 0; n < wPlace.size (); n++) {
      Asteroids AsteroidsPlace = wPlace.get(n);
      if (sq(dist (AsteroidsPlace.getX(), AsteroidsPlace.getY(), myDron.getX(),
myDron.getY())) < sq (4)) {
       AsteroidsPlace.loseFreedom(-1);
       myDron.setState (6);
       myDron.getTotalSt(1);

       println ("gainfreedom");
      }
     }
    }
    myDron.script();
  }


  //deleting Drons, when their health will be equal 0
  for (int i = 0; i < drons.size (); i++) {
    Dron d = (Dron) drons.get(i);
    if (d.isDead()==1) {

     if ( (d.getState() == 2) |(d.getState() == 3) ) {

      for (int n = 0; n < wPlace.size (); n++) {
       Asteroids AsteroidsPlace = wPlace.get(n);
       if (sq(dist (AsteroidsPlace.getX(), AsteroidsPlace.getY(), d.getX(), d.getY())) <
sq (25)) {
        AsteroidsPlace.loseFreedom(-1);
       }
      }
     }

     if ((d.getState() == 12) ) {

      for (int n = 0; n < wPlace.size (); n++) {
       Asteroids AsteroidsPlace = wPlace.get(n);
```

```
        if ( (AsteroidsPlace.getX() == d.NearGetX()) && (AsteroidsPlace.getY() ==
d.NearGetY()) ) {
          AsteroidsPlace.loseFreedom(-1);
        }
       }
     }

     drons.remove(i);
     drons.add(new Dron (MainShip.getX(), MainShip.getY(), random(5, 15),
random(1, 10), random(1, 15) ));
     println("NEW DRON ! ! !");
   }
  }


  //deleting AsteroidsPlaces, when their size will be equal 0
  for (int i = 0; i < wPlace.size (); i++) {
    Asteroids wp = wPlace.get(i);
    if (wp.getSize() < 0.5) {
      wPlace.remove(i);
      //wPlace.add(new Asteroids (random (0, width), random (0, height), random (10,
40)));
    }
  }



  //listing of the asteroids

  int Drx = 50;
  int Dry = 50;
  int c = 0;
  for (Dron dronny : drons) {

    pushStyle();
    fill(0, 0, 255);
    if ( dronny.getTotalSt() >=10) {
      text ("#" + c + " " + "fuel :" + int (dronny.getMaxFuel() ) + " health :" + " " + int
(dronny.getMaxHealth()) + " " +" mass :" + " " +  int (dronny.getMass()), Drx, Dry);
      Dry += 25;
    }
    // Dry += 25;
```

```
    c += 1;

    popStyle();
  }


  //listing of the drons
  int Dx = height - 150;
  int Dy = 50;
  int a = 0;
  for (Dron dronny : drons) {

    pushStyle();
    fill(0, 0, 255);
    text ("#" + a + " " + "x:" + int (dronny.getX()) + " y:" + " " + int (dronny.getY()) + " " +
int (dronny.getTotalSt()), Dx, Dy);
    Dy += 25;
    a += 1;

    popStyle();
  }
}

class Asteroids {

  float x, y, size;
  int isDead = 0;
  float freedom = 3;

  Asteroids(float _x, float _y, float _size) {
    x = _x;
    y = _y;
    size = _size;
  }

  void display() {
    pushStyle();
    strokeWeight (2);
    stroke (#9E2E90);
    fill(#C148B2);
    ellipse(x, y, map (size, 0, 5000, 0, 50), map (size, 0, 5000, 0, 50));
```

```
    fill (#435A63);
    text ("s: " + int(size), x + 15, y);
    text ("fr: " + freedom, x + 15, y + 10);


    textSize(12);
    popStyle();
  }

  void script() {
    display();
  }

  float getFreedom() {
    return freedom;
  }

  void loseFreedom( int fd ) {
    if (freedom >= 0) {
      freedom = freedom - fd;
    }
    if (freedom >= 3) {
      freedom = 3;
    }
    //else freedom = 0;
  }

  float getSize () {
    return size;
  }

  float getX () {
    return x;
  }

  float getY () {
    return y;
  }

  // When dron is collecting resosrses ( water ) - > amount of resourses on it
descreases - > size descreases proportionally amount of res .
  void isBeingEaten (float part) {
```

```
    if (part > size) {
      isDead = 1;
    }
    size = size - part;
  }

  int isDead () {
    return isDead;
  }
}



class Dron {

  //globals for all the functions

  // personal parameters
  PVector currentPlace;
  PVector target;
  PVector location;
  float m;
  float health;
  float size = 6;
  float fuel;
  float TotalStorage = 1;
  float k_f, k_h;
  color few = color(#2B34AA);
  color middling = color(#414474);
  color many = color(#414359);
  color null_ = color(#1523D4);

  float storage = 0;
  float state = 0;
  float maxstorage;
  float waterStartMining = 1000;

  // when death came..
  int isDead = 0;
  int isNotMoving = 0;

  //information about MainShip/Asteroid at that moment
```

```
PVector WaterLocation = new PVector(0, 0);
PVector BaseLocation;
PVector nearestPlace = new PVector (width, height);
float base_dist = 0;
float wat_dist = 0;
float max_dist = 0;

// motion function
PVector speed;
PVector acceleration;
PVector velocity;
float stop_distance = 70;
float maxforce = 2.5;
float maxspeed = 1.5;


float dHealth = 0.01;
float dFuel = 0.01;
float plusFuel = 1;
float part = 0.01;
float maxfuel = 0;
float maxhealth = 0;
float med = 0.1;

Dron ( float tempXpos, float tempYpos, float tempm, float _fuel, float _health) {
  m = tempm;
  fuel = _fuel;
  health =  _health ;
  maxfuel = fuel + 2*m/3;
  maxhealth = health + 2*m;
  location = new PVector (tempXpos, tempYpos);
  speed = new PVector (10, 10);
  target = new PVector (0, 0);
  velocity = new PVector (0, 0);
  acceleration = new PVector (5, 5);
  maxforce = m ;
  maxspeed = m/5;
}


// Am I at my MainBase ?
int atBase() {
```

```
   if (abs(location.x - MainShip.getX()) < 10 && abs(location.y - MainShip.getY()) < 10
&& target.x != 50 && target.y != 50)
     return 1;
   else
     return 0;
  }

  // Am I at my current target ?
  int atTarget() {
   if (abs(location.x - target.x) < 20 && abs(location.y - target.y) < 20 && target.x !=
50 && target.y != 50)
     return 1;
   else
     return 0;
  }

  // drawing out little robots
  void display() {

   if (TotalStorage < 2 ) {
    pushMatrix();
    fill(null_);
    ellipse(location.x, location.y, size + m/4, size + m/4);
    popMatrix();
   }

   if (TotalStorage >= 2 && TotalStorage < 5 ) {
    pushMatrix();
    fill(few);
    ellipse(location.x, location.y, size + m/4, size + m/4);
    popMatrix();
   }

   if (TotalStorage >= 5 && TotalStorage <10 ) {
    pushMatrix();
    fill(middling);
    ellipse(location.x, location.y, size + m/4, size + m/4);
    popMatrix();
   }

   if (TotalStorage >= 10) {
    pushMatrix();
```

```
    fill(many);
    ellipse(location.x, location.y, size + m/4, size + m/4);
    popMatrix();
    k_f = maxfuel / m;
    k_h = maxhealth / m;
  }
 }


 // vital functions
 void life() {
  maxstorage = 100/m;
  fuel = fuel - dFuel;
  if (fuel < 0) {
   isNotMoving = 1;
  }
  if (health < 0) {
   isDead = 1;
  }
  if ( isNotMoving == 1) {
   maxspeed = 0;
   maxforce =0;
   isDead = 1;
  }
 }

 void getTotalSt( int ts ) {
  TotalStorage = TotalStorage + ts;
 }
 // am I still alive ?
 int isDead() {
  return isDead;
 }

 // What shall I do ?
 void changeState() {

  // where is  my MainShip?
  base_dist = dist (location.x, location.y, MainShip.getX(), MainShip.getY());

  /* Well, few words about these states. State - certain condition, depending on
which behavior of drone is determined .
```

```
    state 0 - first dron's condition - does MainShip need me for help in drgging
resourses ?
    state 1 -" oh, atMainShip aren't enough resources. I need to find the closest  free
asteroid "
    state 12(1.2) - I try to find the nearest ast for me.
    state 2 - start collecting resourses. While Dron is collecting resourses, his health
is descreasing too .
    state 3 - Dron's storage is max - Dron's next target - MainShip
    state 4 - Dron is on MainShip, it gives away resourses . Its parametres, like fuel
and health are being increased .
    state 6 - interm
    MainShip doesn't need resorses -> state = 0 .
    */

   if (state == 0 && (MainShip.getWater() < waterStartMining) && (storage < part))
state = 1;
   if (state == 12) {
     setTarget (nearestPlace.x, nearestPlace.y);
   }


   if (state == 12 && (atTarget() == 1) && (storage < maxstorage)) state = 2;

   if (state == 2 ) {


     storage = storage + part;
     health = health - dHealth;
   }

   if ((state == 2) && (storage >= maxstorage)) state = 3;

   if (state == 3) {
     setTarget (MainShip.getX(), MainShip.getY());
   }

   if (state == 6 && (atBase() == 1) && (storage > 0)) state = 4;

   if (state == 4) {

     if (storage > part) {
       storage = storage - part;
```

```
      MainShip.addWater(part);
    }

    if (fuel < maxfuel) {
      fuel = fuel + plusFuel;
    }
    if (health < maxhealth) {
      health = health + med;
    }
  }


  if (storage < part && state == 4 && (health >= maxhealth) && (fuel >= maxfuel)) {
    state = 0;
  }


  if (state == 0) {
    setTarget (MainShip.getX(), MainShip.getY());
    storage = 0;
  }
}

//motion function

void seek(PVector target) {
  PVector desired = PVector.sub (target, location);
  float d = desired.mag();
  desired.normalize();

  if (d < stop_distance) {
    float con = map (d, 0, stop_distance, 0, maxspeed);
    desired.mult (con);
  } else {
    desired.mult(maxspeed);
  }

  PVector steer = PVector.sub(desired, velocity);
  steer.limit (maxforce/m);

  acceleration.add (steer);
}
```

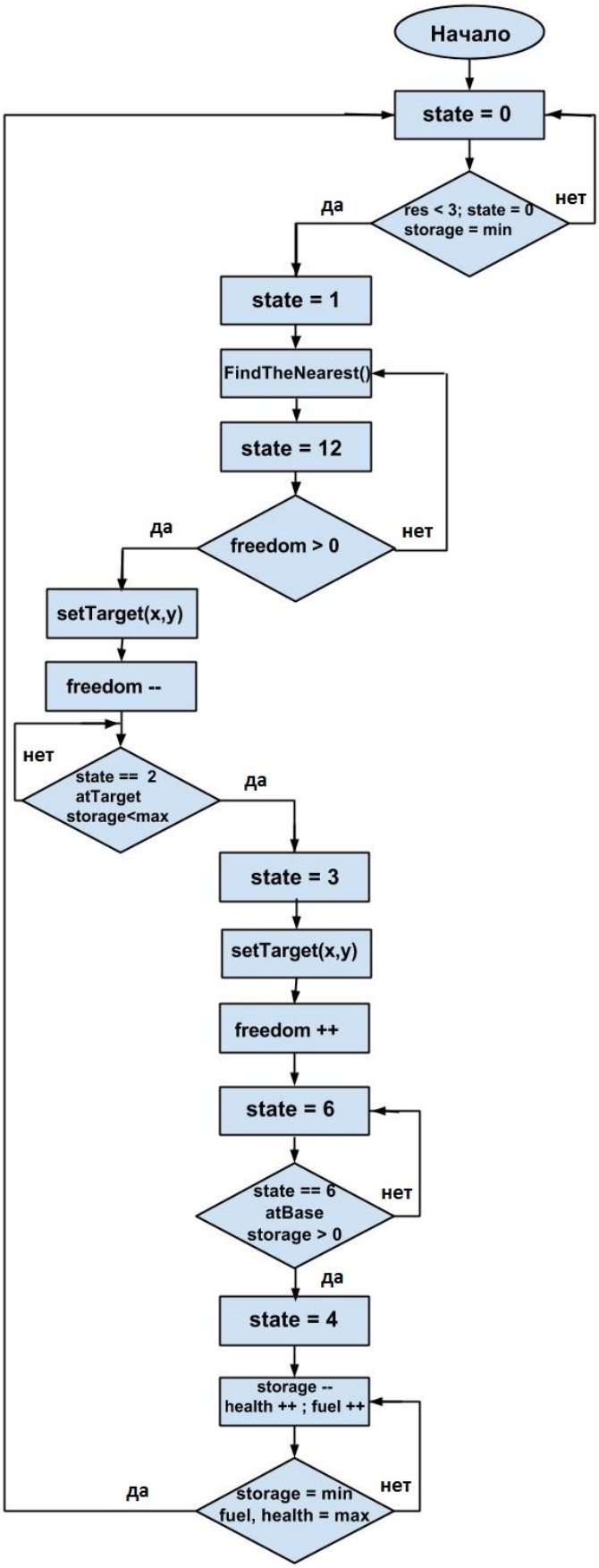
```
void update() {
  velocity.add (acceleration);
  velocity.limit (maxspeed);
  location.add (velocity);
  acceleration.mult(0);

  pushStyle();
  stroke (255, 255, 255);
  strokeWeight(1);
  popStyle();
}

// find the Nearest Asteroid
void LocateWater (int i, float x, float y, float freedom, int isDead) {
  currentPlace = new PVector (x, y);
  wat_dist = dist( location.x, location.y, currentPlace.x, currentPlace.y);

  if (i == 0) {
    max_dist = sqrt (height*height + width*width);
  }

  println(freedom);
  if ((wat_dist < max_dist)&&(freedom > 0)&&(isDead<1)) {
    nearestPlace.set(x, y);
    max_dist = wat_dist;
  }
}

void setTarget (float x, float y) {
  target.x = x;
  target.y = y;
}



void move() {
  seek(target);
  update();
  changeState();
}
```

```
// doing all for this character
void script() {
  life();
  move();
  display();
  changeState();
}


void setState (int st) {
  state = st;
}

float getPart() {
  return part;
}

float getX () {
  return location.x;
}

float getY () {
  return location.y;
}
float getState() {
  return state;
}

float getMaxFuel() {
  return maxfuel;
}

float getMaxHealth() {
  return maxhealth;
}

float getMass() {
  return m;
}
```

```
  float getTotalSt() {
    return TotalStorage;
  }

  float NearGetY () {
    return nearestPlace.y;
  }

  float NearGetX () {
    return nearestPlace.x;
  }
}

// making our main base, which lives its own life .
//our  MainShip
class Base {

  PVector location;
  PVector status;
  // x - ... on our base

  Base () {
    location = new PVector (width/2, height/2);
    status = new PVector (random (0, 10), random (0, 6));
  }


  void display() {
    pushStyle();
    strokeWeight (2.5);
    stroke (#E44642);
    fill(#FF635F);
    ellipse(location.x, location.y, 25, 25);
    popStyle();
  }

  // Base has its total "score", which is constantly diminishing
  void repair() {
    status.x = status.x - 0.01;
    if (status.x < 0) status.x = 0;
  }
```

```
  //updaiting base's total resourses, when bots or drons are on it
  void addRes (float addRes) {
    status.x = status.x + addRes;
  }

  void addWater (float addWater) {
    status.y = status.y + addWater;
  }


  //doing all
  void script() {
    display();
    repair();
  }

  float getX () {
    return location.x;
  }

  float getY () {
    return location.y;
  }

  float getRes() {
    return status.x;
  }
  float getWater () {
    return status.y;
  }
}
```

2) Блок-схема, наглядно показывающая принцип поведения Дрона



Значение переменных в блок-схеме :

- **state** - состояние Дрона в данный момент
- **res** - количество ресурсов на Базе
- **storage** - количество ресурсов, которое может унести Дрон
- **FindTheNearest()** - алгоритм поиска ближнего Астероида
- **freedom** - количество дронов, которое может вместить на себя Астероид
- **setTarget()** - алгоритм установки координат цели
- **fuel** - количество топлива Дрона
- **health** - количество “здоровья” Дрона
- **min, max** - значения минимума/ максимума переменной